CSE 527 Notes: Lecture 13 11/09/2005, More HMMs

Ahrim Youn (based on earlier notes by Raphael Hoffman, Martha Mercaldi, and online resources by Narada Warakagoda)

Hidden Markov Models

Hidden Markov Models (HMM) consist of the following:

States: 1,2, ...

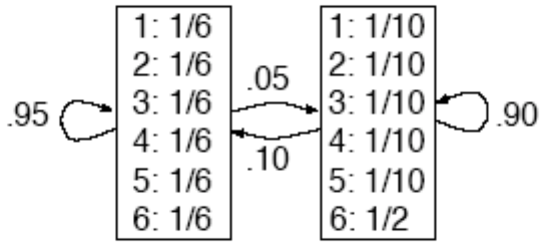
Paths: sequences of states $\pi = (\pi_1, \pi_2, ..., \pi_n)$

Transition probability: $a_{kl} = \Pr(\pi_i = l \mid \pi_{i-1} = k)$

Emission probability: $e_k(b) = \Pr(x_i = b \mid \pi_i = k)$

If we assume that our data was generated by this model, our observed data would be only the emission sequence. Unlike in our previous example, where we identified each state with our current nucleotide (A, C, G or T), we now don't know exactly in what state we are in anymore. Thus, the state/transition sequence can be regarded as hidden data.

Example Let's regard the sequence of rolling a die in a casino. Our opponent cheats from time to time and exchanges the fair die with a loaded die. Our task is to determine the fair die/loaded die sequence by only looking at the sequence of die rolls. Our model could look like this.



Our (observed) emission sequence and (hidden) transition sequence:

Rolls 651166453132651245636664

Die LLLLLLFFFFFFFFFFFFLLLLL

In Computational Biology we are interested if a sequence, e.g. CGCG, came from C+G+C+G+ or C_G_C_G_ or C+G_C+G_ or ... We don't know the state sequence π (hidden data). However, we can calculate the joint probability of a given path π and an emission sequence x:

$$\Pr(x,\pi) = a_{0\pi_i} \prod_{i=1}^{n} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$

Alternative questions that arise in this context are:

1. What is the most probable (single) path in our model, if we are given an emission seq. x? that is, what is π^*?

$$\pi^* = \arg\max_\pi \Pr(x,\pi)$$

2. What is the sequence of most probable states, if we are given an emission sequence x? that is, what is $\hat{\pi}_i$?

$$\hat{\pi}_i = \arg\max_k \Pr(\pi_i = k \mid x)$$

We can solve the first question by applying the Viterbi algorithm.

The Viterbi Algorithm

The Viterbi algorithm computes $\pi^* = \arg\max_\pi \Pr(x,\pi)$. Hence, it is useful if one path dominates all the others. However, if this is not the case, i.e. many good paths are almost equally likely, then $\Pr(x,\pi^*)$ could be very low and other approaches may be preferable. One key problem to finding it is that there are exponentially many paths. However, the Viterbi algorithm is a dynamic programming approach and is computationally efficient.

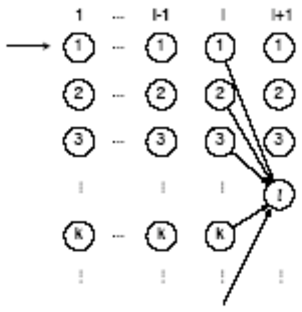
In order to facilitate the computation we define an auxiliary variable,

$$v_l(i) = \max_{\pi_1,\pi_2,\ldots,\pi_{i-1}} \Pr(\pi_1,\pi_2,\ldots,\pi_{i-1},\pi_i = l, x_1, x_2, \ldots, x_i)$$

which gives the probability of the most probable path emitting $x_1, x_2, \ldots, x_i$ and ending in state l. It is easy to observe that the following recursive relationship holds.

$$v_l(i+1) = e_l(x_{i+1}) \cdot \max_k (v_k(i) a_{k,l}) \quad (*)$$

where, $v_l(0) = \begin{cases} 1 & \text{if } l\text{=Begin state} \\ 0 & \text{otherwise} \end{cases}$

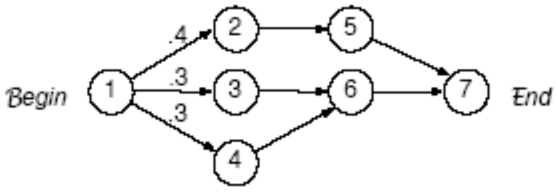


To retrieve the state sequence, we need to keep track of the argument that maximized Eq.(*), for each i and l. We always keep a pointer to the "winning state" in the recursion. Finally the state l^* is found where $l^* = \arg\max_l v_l(n)$ and starting from this state, the sequence of states is back-tracked as the pointer in each state indicates.This gives the required sequence of states.

There is one important issue when implementing this algorithm: The emission probabilities and transition probabilities are typically less than 1. Thus we end up multiplying thousands of fractions which are below 1, which often causes floating point underflows. One solution to this problem is to use logarithms.

Shortcomings of Viterbi

In the following figure, the most probable path goes through 5, but most probable state at 2nd step is 6. (Viterbi is not the only interesting answer.)



Forward and backward algorithm

We want to calculate the probability of the observed sequence, $\Pr(x_1, x_2, ..., x_i)$. We can calculate this quantity using simple probabilistic arguments. But this calculation involves number of operations which is exponential w.r.t i. This is very large even if the length of the sequence, *i* is moderate. Therefore we have to look for an other method for this calculation. Fortunately there exists one which has a considerably low complexity and makes use an auxiliary variable, $f_k(i)$ called *forward variable*.

The forward variable is defined as the probability of the partial observation sequence $x_1, x_2, ..., x_i$ when it terminates at the state k. Mathematically,

$$f_k(i) = \Pr(x_1, x_2, ..., x_i, \pi_i = k)$$

Then it is easy to see that following recursive relationship holds.

$$f_l(i+1) = e_l(x_{i+1}) \sum_k f_k(i) a_{k,l}$$

Using this recursion we can calculate the required probability,

$$\Pr(\vec{x}) = \sum_{\pi} \Pr(\vec{x}, \vec{\pi}) = \sum_k f_k(n) a_{k,0}$$

The complexity of this method, known as the *forward algorithm* is linear w.r.t *i* whereas the direct calculation mentioned earlier, had an exponential complexity.

In a similar way we can define the *backward variable* $b_k(i)$ as the probability of the partial observation sequence $x_{i+1},\ldots,x_n$, given that the current state is *k*. Mathematically ,

$$b_k(i) = \Pr(x_{i+1},\ldots,x_n \mid \pi_i = k)$$

As in the case of $f_k(i)$ there is a recursive relationship which can be used to calculate $b_k(i)$ efficiently.

$$b_k(i) = \sum_l e_l(x_{i+1}) b_l(i+1) a_{k,l}$$

where,

$$b_k(n) = a_{k,0}$$

Further we can see that,

$$\Pr(\pi_i = k \mid \vec{x}) = \frac{\Pr(\pi_i = k, \vec{x})}{\Pr(\vec{x})} = \frac{f_k(i) b_k(i)}{\Pr(\vec{x})}$$

Therefore this gives a way of calculating the posterior probability of a state i being k.

Also this gives another way to calculate $\Pr(\vec{x})$, by using both forward and backward variables :

$$\Pr(\vec{x}) = \sum_k f_k(i) b_k(i)$$

Posterior Decoding

Terminology— posterior: based on probability after seeing data;
prior: based on probability before seeing data

If the most likely path accounts for less than half of the probability, posterior decoding may make sense.

$\hat{\pi}_i = \arg\max_k \Pr(\pi_i = k \mid \bar{x})$

Notice that because transitions are not taken into account, it is possible that this sequence of states isn't even legal in the model; see example under "Viterbi shortcomings" again..

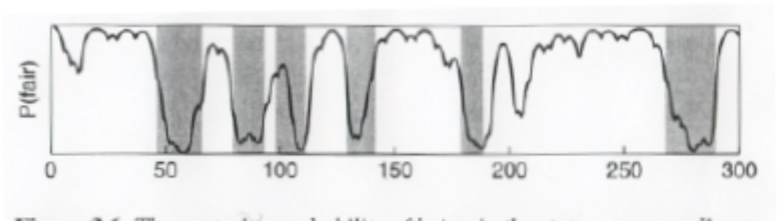


Figure 3.6 *The posterior probability of being in the state corresponding to the fair die in the casino example. The x axis shows the number of the roll. The shaded areas show when the roll was generated by the loaded die.*

Alternative question : given some function g(k) on states, what's its expectation. E.g., the probability of the "+" sub-model in the CpG HMM is E(g(k)) where g(k)=1 iff k is a "+" state.

$$G(i \mid \vec{x}) = \sum_k \Pr(\pi_i = k \mid \vec{x}) \cdot g(k)$$

CpG Example

Data: 41 human sequences, totaling 60 kbp, with 48 CpG islands with an average length of about 1 kbp each.

Viterbi: Found 46 of the 48 islands, plus 121 false positives. With the addition of a post processing step applied to the data which merged any two islands that were within 500bp of each other and then deleted short islands (<500 bp), Viterbi still found 46/48, but with only 67 false positives.

Posterior Decoding: Found the same 46 of the 48 islands and 276 false positives. With the post processed data, posterior decoding still found 46/48 islands, but only 93 false positives. There was little difference between the two methods in this example (with Viterbi being slightly better), but we'll see cases presently where it makes a much bigger difference.

Training of probability parameters

(For now we defer the question of how to decide on the structure of a model, and ask how, given a structure, to identify appropriate parameters.)
Given a model topology and independent training sequences, we want to find emission probabilities for each state and transition probabilities for each transition.
If the path of states through the HMM is known, we can use Maximum Likelihood Estimation (MLE).

$$a_{kl} = \frac{count(k \to l)}{count(k \to anywhere)}$$

$e_k(b)$ = a similar ratio, the fraction of the time that you're in state k that you emit b.
If π is hidden, use Expectation Maximization (EM) to estimate the state sequences and parameters in an iterative loop: Given π estimate the parameters , then with that parameter, estimate π , and repeat until results converge.

If you've got a model with 8 states, for each state you'd need 8 transition probabilities and 4 emission probabilities. These 8*12 parameters are quite many, so even for this small a state diagram we require a fair amount of training data.

Viterbi Training

Make your initial parameter estimates (either randomly or preceded if you have strong prior knowledge of what they might look like). Calculate the Viterbi path for each training sequence and from that count the transitions and emissions, thereby creating a new parameter. Use this new parameter and recalculate the Viterbi path for each training sequence. Iterate until parameter stops changing (or change very little).

Advantages:

- Fast and simple
- Viterbi path is discrete and therefore this algorithm will converge

Disadvantages:

- May converge only on a local optimum, not the global one.
- Not actually maximizing the likelihood we want.

Regarding this last disadvantage, we are looking only at the most probable paths, not at all paths. Also, we are not getting any parameter estimates along paths that do not occur in the training data.